**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**TALLER 11:**

REFRACTORING

**ALUMNO:**

BARRAGAN GUILLIN CRISTHIAN JOSEPH

GUTIERREZ PADILLA VALERIA NICOLE

OWUEN ALEX YAGUAL PANIMBOZA

MALDONADO JARAMILLO PAULETTE YAMILE

**DOCENTE:**

MSIG. DAVID JURADO MOSQUERA

**MATERIA:**

DISEÑO DE SOFTWARE

**AÑO:**

2024 TÉRMINO I

**URL GITHUB:** <https://github.com/xHianx/Taller11-Dise-o.git>

Contenido

[**Enunciado:** 3](#_Toc173958940)

[**Simplifying Method Calls** 4](#_Toc173958941)

[ **Rename method** 4](#_Toc173958942)

[**Consecuencia de mantener el código actual:** 4](#_Toc173958943)

[**Beneficio de refactorizarlo:** 4](#_Toc173958944)

[**Solución en código Java:** 4](#_Toc173958945)

[ **Remove parameter** 5](#_Toc173958946)

[**Consecuencia de mantener el código actual:** 6](#_Toc173958947)

[**Beneficio de refactorizarlo:** 6](#_Toc173958948)

[**Solución en código Java:** 6](#_Toc173958949)

[ **Preserve Whole Object:** 6](#_Toc173958950)

[**Consecuencia de mantener el código actual:** 7](#_Toc173958951)

[**Beneficio de refactorizarlo:** 7](#_Toc173958952)

[**Solución en código Java:** 7](#_Toc173958953)

[ **Parameterize Method:** 8](#_Toc173958954)

[**Consecuencia de mantener el código actual:** 9](#_Toc173958955)

[**Beneficio de refactorizarlo:** 9](#_Toc173958956)

[**Solución en código Java:** 9](#_Toc173958957)

[ **Hide method** 10](#_Toc173958958)

[**Consecuencia de mantener el código actual:** 11](#_Toc173958959)

[**Beneficio de refactorizarlo:** 11](#_Toc173958960)

[**Solución en código Java:** 11](#_Toc173958961)

[**Organizing Data** 12](#_Toc173958962)

[ **Self encapsulate field** 12](#_Toc173958963)

[**Consecuencia de mantener el código actual:** 12](#_Toc173958964)

[**Beneficio de refactorizarlo:** 12](#_Toc173958965)

[**Solución en código Java:** 12](#_Toc173958966)

[ **Replace data value with object** 14](#_Toc173958967)

[**Consecuencia de mantener el código actual:** 14](#_Toc173958968)

[**Beneficio de refactorizarlo:** 14](#_Toc173958969)

[**Solución en código Java:** 14](#_Toc173958970)

[ **Replace magic number with symbolic constant** 16](#_Toc173958971)

[**Consecuencia de mantener el código actual:** 16](#_Toc173958972)

[**Beneficio de refactorizarlo:** 16](#_Toc173958973)

[**Solución en código Java:** 16](#_Toc173958974)

[ **Encapsulate field** 17](#_Toc173958975)

[**Consecuencia de mantener el código actual:** 17](#_Toc173958976)

[**Beneficio de refactorizarlo:** 17](#_Toc173958977)

[**Solución en código Java:** 17](#_Toc173958978)

[ **Encapsulate collection** 18](#_Toc173958979)

[**Consecuencia de mantener el código actual:** 18](#_Toc173958980)

[**Beneficio de refactorizarlo:** 18](#_Toc173958981)

[**Solución en código Java:** 18](#_Toc173958982)

# **Enunciado:**

**Descomprima el archivo del proyecto llamado "TallerRefactoring2.rar", analícelo y comprenda su contexto. Una vez revisado el código fuente, identifique donde se pueden las "técnicas de refactorización" proporcionadas en la parte inferior, luego elabore un informe detallado en el que indique las consecuencias de mantener el código actual, el nombre de la técnica a utilizar y el beneficio de refactorizarlo. Además, proporcione soluciones en código Java para resolver estos problemas.**

# **Simplifying Method Calls**

* **Rename method**

Aplicar la técnica "Rename Method" en el método realizaConsul es crucial para proporcionar un nombre que refleje de manera clara y precisa su propósito y funcionalidad. Un nombre descriptivo mejora la legibilidad del código y facilita la comprensión de su propósito, especialmente para desarrolladores que no estén familiarizados con el código. Esto reduce la ambigüedad y mejora la usabilidad del método.

## **Consecuencia de mantener el código actual:**

Mantener el código con un nombre de método poco claro puede llevar a malentendidos sobre su funcionalidad, dificultando su uso correcto y eficiente. La falta de un nombre descriptivo dificulta la localización y el mantenimiento del método. Los desarrolladores pueden tener dificultades para entender el propósito del método y su implementación, lo que puede llevar a errores y aumentar la complejidad del mantenimiento.

## **Beneficio de refactorizarlo:**

Refactorizar el método realizaConsul para cambiar su nombre a realizarConsulta elimina la ambigüedad y facilita la comprensión de su propósito. Un nombre descriptivo hace que el código sea más legible y fácil de entender, mejorando la mantenibilidad y facilitando la extensión futura del código. Los desarrolladores podrán identificar rápidamente el propósito del método y utilizarlo de manera correcta y eficiente, contribuyendo a un código más limpio y eficiente.

## **Solución en código Java:**

Código inicial:

Captura de pantalla de un celular

Descripción generada automáticamente

Código refactorizado:

Captura de pantalla de un celular

Descripción generada automáticamente

* **Remove parameter**

Aplicar la técnica "Remove Parameter" en el método realizaConsul es crucial para simplificarlo y clarificar su propósito. Al mantener solo los parámetros necesarios, el método se vuelve más fácil de entender y utilizar correctamente. La eliminación de parámetros no utilizados evita confusiones y malentendidos sobre los requisitos del método.

## **Consecuencia de mantener el código actual:**

Mantener el código en su estado actual con parámetros no utilizados aumenta la complejidad y la posibilidad de errores. Los desarrolladores pueden malinterpretar el propósito del método, creyendo que todos los parámetros son necesarios para su ejecución, lo cual no es cierto. Esto puede llevar a un uso incorrecto del método y a dificultades en su mantenimiento y modificación futura.

## **Beneficio de refactorizarlo:**

Refactorizar el método realizaConsul elimina la complejidad innecesaria y hace que el propósito del método sea más claro. Los desarrolladores podrán entender y utilizar el método de manera más precisa, reduciendo el riesgo de errores. Además, el código se vuelve más limpio y eficiente, mejorando la mantenibilidad y facilitando futuras modificaciones.

## **Solución en código Java:**

Código inicial:

Texto

Descripción generada automáticamente

Código refactorizado:

Captura de pantalla de un celular

Descripción generada automáticamente

* **Preserve Whole Object:**

La técnica "Preserve Whole Object" se aplica aquí para simplificar y mejorar la mantenibilidad del código. En el código actual, el método agendarConsulta recibe tanto un objeto Paciente como un objeto Consulta. Sin embargo, Consulta ya contiene un Paciente, lo que hace redundante pasar el Paciente por separado.

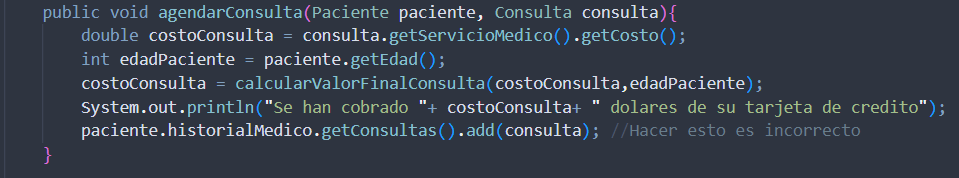
## **Consecuencia de mantener el código actual:**

* El método agendarConsulta recibe dos parámetros donde uno de ellos (el Paciente) ya está contenido dentro del otro (el Consulta). Esto introduce redundancia en la interfaz del método.
* Si en el futuro se necesita cambiar la forma en que se maneja el Paciente dentro de Consulta, se tendrá que revisar y posiblemente cambiar múltiples partes del código donde se pasa tanto Paciente como Consulta.
* El método es más complejo de lo necesario, lo que puede llevar a errores o confusiones al usar el método.

## **Beneficio de refactorizarlo:**

* Al pasar solo el objeto Consulta, el método agendarConsulta se simplifica, haciendo que el método sea más fácil de entender y utilizar.
* La lógica para obtener el Paciente a partir de una Consulta se centraliza dentro del método agendarConsulta. Esto significa que cualquier cambio en cómo se obtiene o se maneja el Paciente solo necesita realizarse en un lugar.
* Menos parámetros significa menos posibilidades de pasar parámetros incorrectos o en el orden incorrecto, lo que reduce el riesgo de errores.
* Si se necesitan agregar más datos en Consulta o cambiar cómo se accede a ellos, solo se necesita modificar la implementación de Consulta y el método agendarConsulta, no todas las llamadas al método.

## **Solución en código Java:**

Código inicial: ****

**Captura de pantalla de computadora

Descripción generada automáticamente**

Código refactorizado:

Texto

Descripción generada automáticamente

**Texto

Descripción generada automáticamente con confianza baja**

* **Parameterize Method:**

Aplicar la técnica "Parameterize Method" en esta parte del código tiene como principal objetivo reducir la duplicación de código y mejorar la mantenibilidad. Actualmente, los métodos realizaConsul y adminMedicamento realizan acciones similares, pero con pequeñas diferencias en sus implementaciones. Al combinar estos métodos en uno solo, utilizamos un parámetro para especificar la acción, lo que centraliza la lógica y facilita su mantenimiento.

## **Consecuencia de mantener el código actual:**

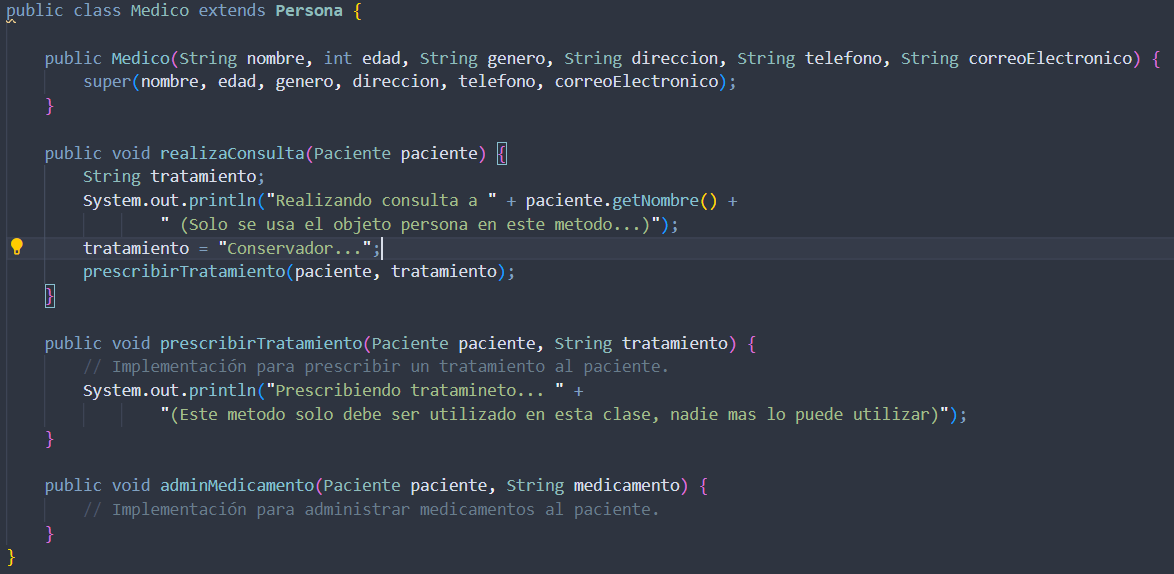
Mantener el código en su estado actual implica mantener métodos duplicados que realizan acciones similares, lo cual incrementa la posibilidad de errores y dificulta el mantenimiento. Cada vez que se necesite agregar una nueva acción o modificar una existente, se deberá crear o modificar múltiples métodos, lo que aumenta la complejidad y el riesgo de inconsistencias.

## **Beneficio de refactorizarlo:**

Refactorizar el código utilizando la técnica "Parameterize Method" simplifica la estructura del código al centralizar la lógica de las diferentes acciones en un solo método. Esto no solo reduce la duplicación de código, sino que también facilita la extensión futura, ya que agregar nuevas acciones es tan sencillo como pasar un nuevo valor de parámetro al método existente. Además, mejora la claridad del código, haciendo que sea más fácil de entender y mantener a largo plazo.

## **Solución en código Java:**

Código inicial:



Código refactorizado:

**Texto

Descripción generada automáticamente**

* **Hide method**

Aplicar la técnica "Hide Method" en el método calcularValorFinalConsulta es crucial para mejorar el encapsulamiento y proteger la lógica interna de la clase. Al cambiar el modificador de acceso a privado, se asegura que el método solo sea accesible dentro de la clase en la que está definido, evitando así el acceso externo no deseado. Esto contribuye a un diseño más modular y robusto, promoviendo un sistema más mantenible y menos propenso a errores.

## **Consecuencia de mantener el código actual:**

Mantener el código con el método calcularValorFinalConsulta como público debilita el encapsulamiento y permite que otras partes del código dependan de su lógica interna. Esto puede llevar a un acoplamiento innecesario y dificultar cambios futuros en la implementación. Además, al ser público, el método puede ser utilizado incorrectamente por otros desarrolladores que no comprendan completamente su propósito, lo que puede llevar a errores y malentendidos.

## **Beneficio de refactorizarlo:**

Refactorizar el método calcularValorFinalConsulta para cambiar su modificador de acceso a privado mejora el encapsulamiento y protege la lógica interna de la clase. Esto reduce el riesgo de que otros componentes del sistema utilicen incorrectamente o dependan de la implementación del método. Al mantener el método oculto, se minimiza el acoplamiento y se facilita el mantenimiento del código, permitiendo cambios en la implementación sin afectar otras partes del sistema. Esta práctica contribuye a un diseño más modular y robusto, promoviendo un sistema más mantenible y menos propenso a errores.

## **Solución en código Java:**

Código inicial:

Texto

Descripción generada automáticamente

Código refactorizado:

Texto

Descripción generada automáticamente

# **Organizing Data**

* **Self encapsulate field**

Aplicar la técnica "Self Encapsulate Field" en la clase Consulta permite mayor flexibilidad en el manejo de los campos privados de la clase. Utilizar getters y setters en lugar de acceso directo a los campos dentro de la clase facilita la implementación de lógica adicional, como la validación de datos o la inicialización diferida, y permite que las subclases redefinan estos métodos según sea necesario.

## **Consecuencia de mantener el código actual:**

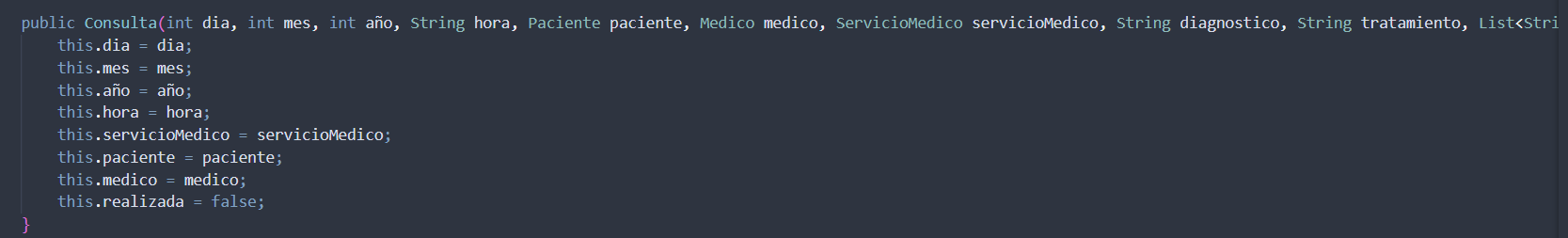
Mantener el acceso directo a los campos privados limita la flexibilidad y la capacidad de extender o modificar la lógica de manejo de datos sin cambiar el código existente. Esto puede llevar a una menor reutilización del código y dificultar la implementación de cambios futuros, como la adición de validaciones o transformaciones de datos.

## **Beneficio de refactorizarlo:**

Refactorizar la clase Consulta para usar getters y setters en lugar de acceso directo a los campos privados mejora la encapsulación y la flexibilidad del código. Permite la implementación de lógica adicional en los métodos de acceso, facilita la extensión de la funcionalidad en subclases y mejora la mantenibilidad del código a largo plazo.

## **Solución en código Java:**

Código inicial:



Texto

Descripción generada automáticamente

Código refactorizado:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* **Replace data value with object**

Aplicar la técnica "Replace Data Value with Object" en el campo costo de la clase ServicioMedico permite encapsular la validación y los comportamientos relacionados con el costo en una clase separada (Costo). Esto facilita la adición de más comportamientos y validaciones en el futuro sin modificar la clase ServicioMedico.

## **Consecuencia de mantener el código actual:**

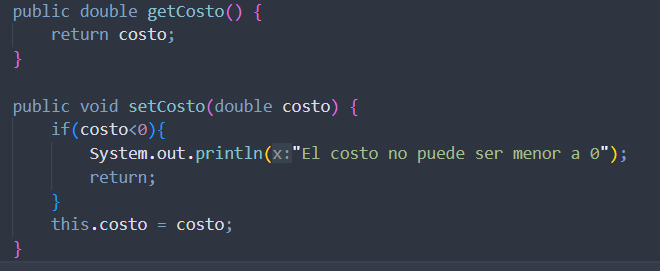
Mantener el campo costo como un tipo primitivo limita la capacidad de agregar comportamientos adicionales y validaciones sin agregar complejidad a la clase ServicioMedico. Esto puede llevar a una menor reutilización del código y dificultar la implementación de cambios futuros.

## **Beneficio de refactorizarlo:**

Refactorizar el campo costo para usar una nueva clase Costo mejora la encapsulación y proporciona un lugar centralizado para agregar comportamientos adicionales y validaciones relacionadas con el costo. Esto mejora la mantenibilidad y la claridad del código a largo plazo.

## **Solución en código Java:**

Código inicial:



Texto

Descripción generada automáticamente

Código refactorizado:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* **Replace magic number with symbolic constant**

Aplicar la técnica "Replace Magic Number with Symbolic Constant" en el número 0.25 que representa el descuento para adultos mayores mejora la claridad del código y facilita su mantenimiento. Usar una constante simbólica hace que el significado del número sea explícito y reduce la posibilidad de errores al modificar el valor.

## **Consecuencia de mantener el código actual:**

Mantener el número mágico 0.25 directamente en el código puede llevar a confusión sobre su significado y propósito. Además, si se necesita cambiar el valor, sería necesario buscar y reemplazar manualmente cada instancia del número, lo que puede resultar en errores.

## **Beneficio de refactorizarlo:**

Refactorizar el número mágico 0.25 para usar una constante simbólica DESCUENTO\_ADULTO\_MAYOR mejora la legibilidad y mantenibilidad del código. La constante sirve como documentación en vivo del significado del valor, y cambiar el valor de la constante se hace de manera centralizada y segura.

## **Solución en código Java:**

Código inicial:

Texto

Descripción generada automáticamente

Código refactorizado:

Texto

Descripción generada automáticamente

* **Encapsulate field**

Aplicar la técnica "Encapsulate Field" en el campo historialMedico de la clase Paciente mejora la encapsulación y permite agregar comportamientos y validaciones adicionales en los métodos getter y setter. Esto facilita el mantenimiento y desarrollo del código, asegurando que los datos se manejen de manera controlada.

## **Consecuencia de mantener el código actual:**

Mantener los campos como públicos compromete la encapsulación, permitiendo que otras partes del código modifiquen los datos directamente sin control, lo que puede llevar a inconsistencias y errores difíciles de rastrear.

## **Beneficio de refactorizarlo:**

Refactorizar los campos públicos para que sean privados y proporcionar métodos getter y setter mejora la encapsulación, facilita la adición de validaciones y comportamientos adicionales, y mejora la mantenibilidad del código. Esto asegura que los datos se manejen de manera controlada y consistente.

## **Solución en código Java:**

Código inicial:

Texto

Descripción generada automáticamente

Código refactorizado:

Texto

Descripción generada automáticamente

* **Encapsulate collection**

Aplicar la técnica "Encapsulate Collection" en las colecciones consultas y recetasMedicas en la clase HistorialMedico mejora la encapsulación y permite un control más estricto sobre la modificación de estas colecciones. Los métodos para agregar y eliminar elementos proporcionan un punto centralizado para la manipulación de la colección, mejorando la mantenibilidad y reduciendo el riesgo de cambios accidentales.

## **Consecuencia de mantener el código actual:**

Mantener las colecciones como accesibles directamente a través de los getters y setters permite que el código cliente modifique las colecciones directamente, lo que puede llevar a inconsistencias y errores difíciles de rastrear.

## **Beneficio de refactorizarlo:**

Refactorizar las colecciones para que sean encapsuladas dentro de la clase y accesibles solo a través de métodos específicos mejora la encapsulación y la integridad de los datos. Los métodos para agregar y eliminar elementos proporcionan un control centralizado y seguro sobre la modificación de las colecciones.

## **Solución en código Java:**

Código inicial:

Texto

Descripción generada automáticamente

Código refactorizado:

Texto

Descripción generada automáticamente